Software Engineering Institute

T-Check in Technologies for Interoperability: Web Services and Security—Single Sign-On

Lutz Wrage
Soumya Simanta
Grace A. Lewis
Saul Jaspan

December 2007

TECHNICAL NOTE
CMU/SEI-2008-TN-026

Integration of Software-Intensive Systems (ISIS) Initiative

Carnegie Mellon

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

List of Figures

List of Tables

Abstract

A single sign-on (SSO) solution is intended to provide a single authentication point for a set of Web services. The SSO solution forwards the necessary authentication information to the Web services, which in turn authenticate the end user to legacy systems that implement the Web services' functionality. This technical note presents the results of applying the T-Check approach in an initial investigation of two Web services standards, WS-Security and SAML, to create an SSO solution that works inside a single organization. This approach involves (1) formulating hypotheses about the technology and (2) examining these hypotheses against specific criteria through hands-on experimentation. The outcome of this two-stage approach is that the hypotheses are either fully or partially sustained or refuted. In this report, four hypotheses—based on claims found in experience reports and on vendor Web sites—are examined: (1) it is possible to implement SSO for the two Web services using SAML and WS-Security; (2) it is fairly easy to implement a basic SSO solution; (3) the SSO solution will not have a major impact on the runtime behavior of the system; and (4) the SSO solution can provide the required access control. The first three hypotheses were sustained; it was not necessary to implement the fourth one to list options for adding access control.

1 Introduction

Single sign-on (SSO) is a method of access control that enables a user to authenticate once for access to the resources of multiple software systems [Wikimedia 2006b]. In a Web services environment, thus, a service requester is authenticated once and gets access to multiple Web services.

A T-Check[SM] investigation is a simple and cost-efficient way to understand what a technology can and cannot do in a specific context [Lewis 2005]. In this T-Check investigation, we explore some of the fundamental technologies and standards for the implementation of Web services SSO. Specifically, this T-Check investigation focuses on finding initial answers to the following questions:

1. Which standards can be used to provide SSO for Web services?
2. What is the effort required to implement an SSO solution?
3. What is the runtime overhead of using an SSO solution?

1.1 WEB SERVICES

A Web service has been defined by the World Wide Web consortium as follows [W3C 2004]:

> *… a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*

Web services are one approach to implementing service-oriented architecture (SOA), where the following conditions apply:

- Service interfaces are described using Web Services Description Language (WSDL) [W3C 2005].
- Message payload is transmitted using Simple Object Access Protocol (SOAP) over HTTP (Hypertext Transfer Protocol) [W3C 2003].
- Universal Description Discovery and Integration (UDDI) is used for service discovery [OASIS 2005]. Its use is optional.

Other combinations of technologies can be used to implement SOA, but using Web services is by far the most common approach. For this reason, the acronym SOA is often used to imply the use of Web services as the implementation technology. For a T-Check investigation of Web services see *Model Problems*[1] *in Technologies for Interoperability: Web Services* [Lewis 2006].

[1] The T-Check approach was called the model problem approach previously and is referred to as such in other Carnegie Mellon® Software Engineering Institute (SEI) technical notes and reports. (Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.)

1.2 SECURITY

Security in a Web services environment has many aspects, such as

- authentication to ensure that a user or other entity is really who it claims to be
- authorization where a Web services consumer can access only permitted subsets of data and functionality provided by a Web service
- confidentiality so that information exchanged between Web services is not available to third parties
- integrity so that it is possible to detect unauthorized modifications to information exchanged between Web services
- privacy, which involves special protection measures for personally identifiable data

In the context of Web services SSO, we are concerned with authentication and authorization, primarily. Authentication verifies the claimed identity of a Web services consumer. The initial authentication can be achieved if an end user or other entity (the subject) provides a set of credentials to an application (e.g., in the form of a user name and password or a smart card). Once the identity of a subject has been confirmed, the application can grant access to those resources that the subject is allowed to use.

After the initial authentication step, the application may invoke Web services to implement the application's functionality. Some of these Web services require information about the consumer, the application's end user, or the organization running this application. This information can, for example, be used for billing or auditing purposes. Since the basic Web services protocols do not explicitly include this kind of information, the required identity information must be included in the messages that applications exchange with Web services. This exchange must be done in a manner that allows the service provider to verify that the provided identity information is trustworthy.

Authorization grants or denies access to resources and services depending on which subject is requesting the access. To do this, authorization relies on authentication to distinguish "good" identities from intruders. After all, it makes little sense to check for a user's permissions before the user's identity is verified. Other aspects also rely on authentication. For example, confidentiality relies on the ability to identify recipients of information, and integrity relies on reliable identification of the sender of information.

For more detail on security in a Web services context, see *Securing Web Services with WS-Security: Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption* [Rosenberg 2004].

1.3 SINGLE SIGN-ON

One reason for the popularity of Web services is that this technology can help to leverage legacy applications in a services-based environment. Many organizations have legacy software systems that are difficult to integrate with other applications because they were developed as standalone applications. Also, it is difficult to make part of a legacy system's functionality available to exter-

nal partners. Web services offer a solution to these problems: create an adapter component that provides externally visible interfaces and exposes them as a Web services [IBM 2002].

However, this approach can cause an authentication problem. What do you do if two systems have separate databases of users and passwords? End users of an application that uses the Web services created for the two legacy systems would have to log in twice, maybe with different user name/password combinations. The goal of SSO is to avoid this inconvenience. With SSO, a user logs in once to gain access to both Web services.

Implementing SSO in a Web services environment is not trivial. There are complications in three areas:

1. choosing the SSO approach

   Many standards and specifications exist, and which ones are needed to provide the desired functionality is not always obvious. Further, many Web services standards are new or under development, which makes it hard to determine whether the selected standards contradict one another or can be used in combination. We focus on this issue in our T-Check investigation.

2. designing and implementing user account management

   The data in user databases maintained by separate legacy systems might overlap. Thus, it is necessary to decide on a global management scheme for user data. Any account management approach chosen may require modification to the legacy systems. This issue is outside the scope of the current T-Check investigation.

3. authentication for composed services

   When a user logs into one of the Web services, that login information has to be passed to other Web services participating in the SSO solution. This forwarding of authentication data is a core technical issue of any SSO approach, and we address it in our investigation.

In a distributed, services-based environment, authentication also includes transfer of authentication-related data between services. If a subject has been authenticated once, reauthentication for each service invocation should not be necessary, even when service calls move across organizational boundaries. A practical SSO solution needs to integrate authentication data across organizations without requiring a central authentication authority. Each organization must be able to manage its own authentication policies and procedures, and inter-organizational recognition of identities and authentication must be governed by contractual agreements.

A technical SSO solution that works across organizations involves federation, the interoperability of authentication and identity data, and the mechanisms that enable the exchange of this data. More formally, federation is defined as follows [Rouault 2005]:

> *Federation is the combination of business and technology practices to enable identities to span systems, networks, and domains in a secure and trustworthy fashion. This is analogous to how passports are used to assert our identity as we travel between countries. An important thing to note is that these domains may exist both within and between enterprises. The main purpose of federation is to share identity information across heterogeneous systems and identity platforms.*

In this T-Check investigation, we explore the Security Assertion Markup Language (SAML) and Web Services Security (WS-Security), two fundamental technologies that form a basis on which an advanced authentication solution including federation can be built. In Section 2, we provide an overview of the various standards that relate to SSO. In Section 3, we describe the T-Check process and how we applied it to SSO. In Section 4, we offer an evaluation of the results of the T-Check investigation. Finally, in the last section, we reflect on lessons learned and the current maturity of SSO technology for Web services.

2 Web Services Security Specifications

The basic Web services standards for message exchange and service descriptions, SOAP and WSDL, do not provide for security. Most Web service implementations use HTTP as the underlying message transport protocol, where the content of an HTTP request/response is a SOAP message. HTTPS provides a measure of security; it encrypts the communication between the computer that requests the service by sending a SOAP request (the HTTP client) and the computer that executes the service and sends a SOAP response (the HTTP server). HTTPS also authenticates the HTTP server, and it can be used to authenticate the HTTP client. However, HTTPS security is limited to a single communication link. In a Web services environment, it is possible that a SOAP message can be processed by one or more intermediary nodes before it reaches the ultimate receiver that processes the service request. Therefore, it is necessary to introduce additional mechanisms to ensure end-to-end security.

Initially, companies responded to this need by creating proprietary security solutions that were not interoperable. The Organization for the Advancement of Structured Information Standards (OASIS), an industry consortium for e-business, saw the need to standardize the way that security for Web services was implemented. In November 2002, OASIS adopted the Security Assertion Markup Language (SAML) 1.0, an XML-based standard for exchanging authentication and authorization data [Wikimedia 2006a]. An updated version (SAML 1.1) was adopted in 2003 to provide clarifications and minor improvements [OASIS 2006b]. For this T-Check, we used an implementation of SAML 1.1.

In a parallel effort to OASIS, a number of organizations developed Web Services Security (WS-Security) 1.0, which is a standard for the communication protocols that secure Web services can use for message exchange [IBM 2004]. WS-Security 1.0 was adopted by OASIS in April of 2002, and effectively filled some of the gaps in SAML. In response to additional needs of Web service developers, WS-Security 1.1 was created and accepted by OASIS in February of 2006 [OASIS 2006a]. In particular, the WS-Security SAML token profile defines a standard way to insert SAML tokens into the header of a SOAP message, making it possible to use SAML in a Web services environment.

Before describing the T-Check approach, we provide a detailed view of the technologies we are examining and an overview of other standards for Web services security.

2.1 SECURITY ASSERTION MARKUP LANGUAGE (SAML)

SAML provides an extensible set of data formats to communicate identity and authentication information in a variety of environments including Web services. It is built around the idea of identity federation, using information from multiple, independently administered sources for identity information to implement authentication and authorization [Lockhart 2005].

The most prominent problem that SAML tries to solve is SSO [Wikimedia 2006a]. SAML attempts to achieve SSO in a general sense by specifying ways to communicate identity information that is crucial when sharing sign-on information. The specification documents also describe how

to use these general message exchanges in the implementation of particular SSO scenarios. These scenarios, however, are limited to Web applications and are not directly applicable to Web services.[2]

SAML defines a number of message formats for sending, receiving, and sharing identity-related information. All these messages are defined as XML documents, which makes it easy to integrate them with other XML formats. Specifically, SAML [Lockhart 2005]

- provides XML formats for user identity information and for the requesting and sending of this identity information
- defines how these messages can be exchanged using the SOAP protocol
- supports a number of privacy protection mechanisms, such as the ability to determine user attributes without revealing user identities
- defines how to use authentication methods from existing, widely used solutions, such as X.509 and PGP public keys, Kerberos tokens, and hardware tokens

Because we used SAML in our T-Check examination, we adopted some of that standard's terminology in this report. A *subject* is any entity (human or computer) that has an identity in a security domain. An *SAML Authority* (SA) has access to identity information about subjects and can verify subjects' credentials. If a subject wants to authenticate itself, it needs to present credentials to an SA. Such credentials can take many forms, including username/password combination or a smart card. An SA makes identity information available in the form of assertions about a subject; the SA *issues* SAML assertions. We call a system that depends on an SA for identity information a *relying party* (RP).

SAML assertions are the main carriers of identity information, so we describe them in some detail. An assertion may contain the following information:

1. the SAML version number
2. a globally unique identifier for the assertion
3. the name of the SA that issued this assertion
4. the time the assertion was issued
5. conditions that constrain the use of an assertion (e.g., a time interval in which the assertion is valid) (optional)
6. an XML Signature that authenticates the assertion (optional)
7. one or more SAML statements
    a. The most important SAML statements are authentication and attribute statements, and for this T-Check investigation, they are the only relevant types. An *authentication statement* reports that the statement's subject was authenticated using a particular method at a particular time. SAML defines the details of more than 20 different authentication methods. An authentication statement must specify its subject by supplying data that allows the subject to be authorized. This may be the subject's X.509 certifi-

[2] A Web application is typically used by a human via a browser-based user interface, whereas a Web service is used by computer programs, including Web applications.

cate. An *attribute statement* contains properties associated with the subject. Typical attributes are the groups or roles of a subject.

b. Other kinds of statements are subject statements and authorization decision statements. A *subject statement* contains information about a subject that may or may not be authenticated; an *authorization decision statement* indicates if a subject is authorized to access a certain resource.

In addition to assertions, SAML also defines a set of messages that can be used in a request-response protocol. An RP can send a request for identity information related to an SA. The request may be for information about the subject, authentication, attributes, or authorization. The SA responds with an assertion containing the requested information.

SAML alone is not sufficient to provide authentication in a Web services environment, because it only specifies the XML schemas for the format of the exchanged XML messages and describes how to construct SOAP messages from those schemas; it does not specify how to integrate SAML with the messages exchanged during Web service interactions. SAML describes how an SAML message is inserted into the body of a SOAP message. In a Web services environment, however, the SOAP body is used for the documents exchanged between a service and its clients, such that SAML tokens should be placed in the header of the SOAP message.

2.2 WEB SERVICES SECURITY (WS-SECURITY)

WS-Security defines how to extend SOAP messages to enable secure Web services. Figure 1 shows how WS-Security modifies a SOAP message.

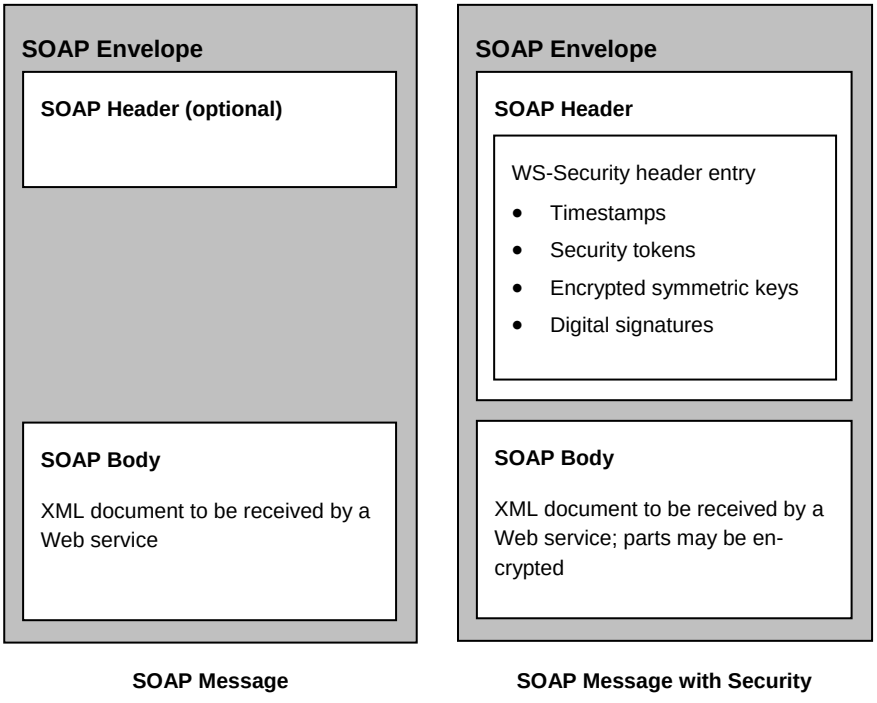

Figure 1: Structure of a SOAP Message

In general, every SOAP message consists of a SOAP envelope containing a header and a body. The header can contain multiple header entries. A SOAP header need only be inserted if it contains at least one header entry; so in the simplest case, there is only a SOAP body that contains the message payload.

To enable security for SOAP messages, WS-Security defines how to encrypt and digitally sign parts of the message. Encryption and digital signatures can be applied to a header entry, to the body, or to part of the body. Also, WS-Security defines how to add timestamps and security tokens to a message. (In Section 5, we describe how we make use of this feature to insert an SAML assertion as a token into SOAP messages that invoke a Web service.) This additional information is contained in a security header entry that is placed in the SOAP header. WS-Security also modifies the SOAP body and other header elements by inserting identifiers needed to reference parts that are encrypted or signed.

A header entry can be processed by the ultimate receiver of the message or by an intermediary node. It is also possible for a SOAP message to contain several header entries, each of which may be processed by a different intermediary node. This feature makes it possible to put specialized processing nodes for certain kinds of message header entries in a network and route SOAP messages through the applicable nodes. A simple example is to have a dedicated node for security processing in a corporate network. Such a node receives all SOAP messages containing WS-Security information, verifies digital signatures, decrypts the message body, and processes SAML tokens. The result of processing the security header entry in that way is a plain SOAP message. The message can then be forwarded to the actual Web service for final processing.

Intermediary SOAP processing nodes are used, for example, in enterprise service buses. Also, vendors are beginning to make such nodes available as appliances.[3]

2.3 OTHER RELATED STANDARDS

Along with the base WS-Security standard, there are some pertinent add-on standards for Web services security:

- **WS-Trust** builds on WS-Security to add the ability to establish, assess the presence of, and broker trust relationships. It also defines methods for issuing, renewing, and validating security tokens. These methods become particularly important if service consumers and providers reside within different trust domains. As an example, assume that a Web services client retrieves an SAML token from an SA and embeds the token into a SOAP message. For the receiver to trust the token, it must first establish trust with the SA. WS-Trust is an OASIS Standard [OASIS 2007c].

[3] An *appliance* is a computer that is pre-configured to provide a specialized function (e.g., indexing a Web site and providing a search function).

- **WS-SecurityPolicy** builds on WS-Security to add the ability to describe how senders and receivers can specify their requirements and constraints in the form of policy assertions. WS-SecurityPolicy is currently available as an OASIS committee specification [OASIS 2007b].
- **WS-Privacy** builds on WS-Security to add the ability to state privacy policies and require that incoming requests make claims about the sender's adherence to these policies. WS-Privacy has been proposed, and there is currently no draft specification publicly available [Microsoft 2002].
- **WS-SecureConversation** builds on WS-Trust and WS-Security to add the ability to establish security contexts between Web services and their clients. A security context can span a series of message exchanges, allowing the creation of an authenticated session. WS-SecureConversation is an OASIS standard [OASIS 2007a].
- **WS-Federation** builds on all the preceding WS-Security standards to define how to construct fully federated trust scenarios. A draft of this standard is publicly available [IBM 2006].
- **WS-Authorization** builds on WS-Trust to add the ability to describe how access policies for a Web service are specified and managed. In particular, it describes how claims may be specified within security tokens and how these claims may be interpreted at the endpoint. WS-Authorization has been proposed, and there is no draft publicly available yet [Microsoft 2002].

Due to the naming scheme, these standards are often referred to as WS-* standards. See Figure 2 for a diagram of how these specifications relate to each other. Standards in one layer are designed to work together, and each layer depends on standards in the next lower layer. Note that some of these standards are still under development.

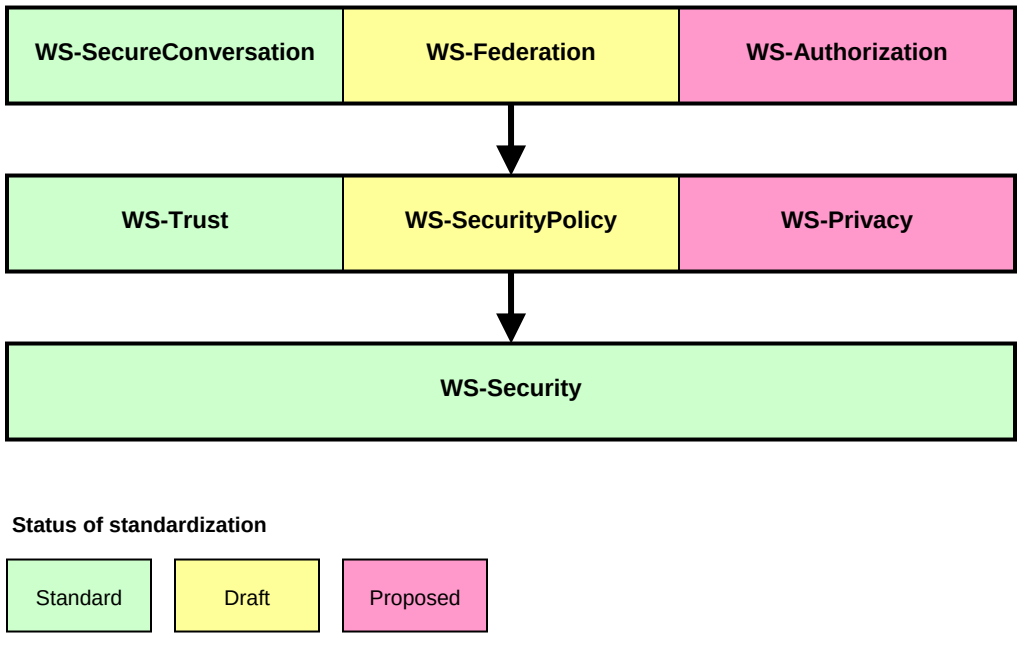

Figure 2: Relationships between WS- Standards and Specifications*

In an independent effort, the Liberty Alliance has developed a set of specifications for federated identity and identity-based Web services [Liberty 2007].

- Liberty Identity Federation Framework (ID-FF) contains specifications related to identity federation and management.
- Liberty Identity Services Interface Specifications (ID-SIS) contains specifications for enabling interoperable identity services.
- Liberty Identity Web Services Framework (ID-WSF) contains specifications providing a framework for building interoperable, identity-based Web services.

Overall, the Liberty Alliance specifications describe capabilities that are similar to WS-Federation and other WS-* standards.

3 Using the T-Check Approach

The T-Check approach is a technique for evaluating technologies. This approach involves (1) formulating hypotheses about the technology and (2) examining these hypotheses against specific criteria through hands-on experimentation. The outcome of this two-stage approach is that the hypotheses are either sustained (fully or partially) or refuted. The T-Check approach has the advantage of producing very efficient and representative experiments that not only evaluate technologies in the context of their intended use but also generate hands-on competence with the technologies [Wallnau 2001]. A graphical representation of the T-Check process is shown in Figure 3.

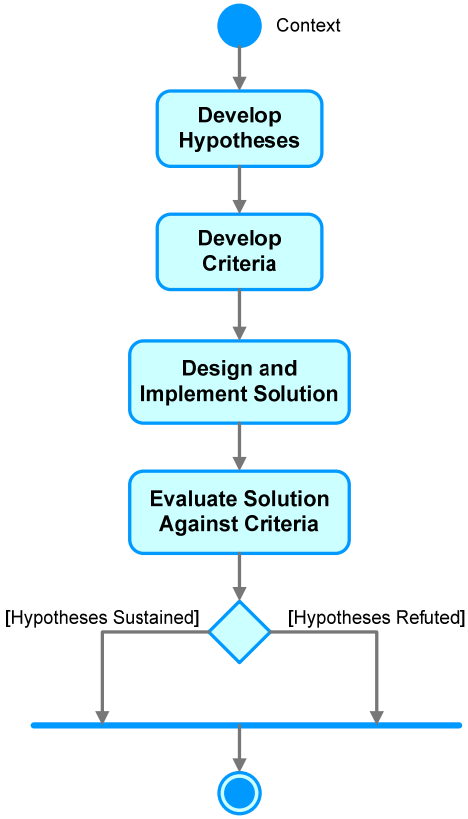

Figure 3: T-Check Process for Technology Evaluation

The T-Check approach is part of a larger process for context-based technology evaluation. In this larger process, the context for the T-Check is established and the expectations from the technology are captured [Lewis 2005].

3.1 T-CHECK CONTEXT

The context for this T-Check investigation is an organization that has two legacy software systems, A and B, whose functionality it wants to make available as Web services. Both systems implement their own authentication; that is, each has a separate and distinct set of usernames and

passwords, and a user needs to log in to each system separately. All users authorized to use system A also need access to system B. Some users only need access to system B.

Only a subset of the functions performed by the legacy systems needs to be made available as services. Each of these functions takes between 15 and 35 seconds to execute.

The organization wants to know

1. Which combination of technologies should be chosen to implement SSO?
2. How much effort will it take to develop an SSO solution?
3. What is the impact on execution time of the SSO solution?
4. How can access control be realized when using SSO?

3.2 DEVELOP HYPOTHESES

For Web services, we defined the following initial hypotheses based on claims found in experience reports and on vendor Web sites:

1. It is possible to implement SSO for the two Web services using SAML and WS-Security.
2. It is fairly easy to implement a basic SSO solution.
3. The SSO solution will not have a major impact on the runtime behavior of the system.
4. The SSO solution can provide the required access control.

3.3 DEVELOP CRITERIA

These are the defined evaluation criteria for the above hypotheses:

Table 1: Evaluation Criteria

Hypothesis	Criteria
It is possible to implement SSO for the two Web services using SAML and WS-Security.	SAML and WS-Security are compatible and can be used together and in combination with other WS-* specifications. There are tools or libraries available that support integrated use of SAML and WS-Security.
It is fairly easy to implement a basic SSO solution.	Once Web services have been created and deployed. It takes no more than 20 person-hours of effort to integrate a basic SSO solution that uses SAML and WS-Security.
The SSO solution will not have a major impact on the runtime behavior of the system.	The overhead introduced by the SSO solution is less than 250 ms per Web service invocation, negligible compared to the overall service execution time.
The SSO solution can provide the required access control.	User authentication data can be extended to include information about user permissions.

3.4 DESIGN AND IMPLEMENT SOLUTION

To evaluate the first hypothesis, we reviewed standards documents, literature, and tool documentation. To evaluate the second and third hypotheses, we implemented a simple Web service and

added SAML-based authentication to it. It was not necessary to implement a solution for the fourth hypothesis; based on the experience we gained while implementing the basic authentication scheme, we were able to list options for adding access control.

4 Assessing the Compatibility of SAML and WS-Security

To determine if the SAML and WS-Security standards are compatible or contradictory we

1. searched in vendor literature for claims about compatibility
2. researched the standards' specifications to see if they are compatible or contradictory
3. looked for tools and libraries that implement both standards, because that would show they are compatible

During a quick search, we found that the WS-Security 1.1 OASIS standard includes an SAML Token Profile [OASIS 2006c], a document that describes how to use SAML assertions as security tokens in WS-Security SOAP messages. We discovered two libraries (Apache WSS4J and Oracle Phaos) whose documentation states that they support WS-Security in combination with SAML tokens. There is also tool and library support for a combination of SAML and WS-Security, as shown in Table 2.

Table 2: Some SSO Tools and Libraries

Tool or Library Name	Specification	Overview	Reference Citation
SAML 1.1 Java Toolkit	SAML	Ping Identity's SAML-1.1 implementation	[SourceID 2006]
OpenSAML	SAML	An open source implementation of SAML 1.1 and 2.0	[Internet2 2007]
WS-Federation for Apache 2.0 Toolkit	WS-Federation	An open source module that extends Microsoft's Active Directory Federation Services (ADFS) and WS-Federation to provide Web SSO for Apache Web applications written in Java, Perl, and PHP	[SourceID 2007]
Apache WSS4J	WS-Security with SAML Tokens	An implementation of the OASIS Web Services Security (WS-Security) from OASIS Web Services Security TC	[Apache 2006]
Oracle Phaos	WS-Security and SAML	Oracle Phaos products provide tools for identity management security and standards-based cryptographic protocols. Components include encryption support, certificate management, secure messaging, secure communications, XML encryption and digital signature, and secure federation.	[Oracle 2007]
DirectControl	WS-Federation	Centrify DirectControl extends ADFS to Web applications running on non-Microsoft platforms.	[Centrify 2007]
ADFS	WS-Federation	ADFS is based on the emerging, industry-supported Web services architecture, which is defined in WS-* specifications.	[Microsoft 2007]

The relationships described in Table 2 are a strong indication that the specifications are indeed compatible.

5 Designing and Implementing the Solution

5.1 DEFINING A SYSTEM ARCHITECTURE BASED ON THE T-CHECK CONTEXT

To design the solution, we first created a notional architecture of the system based on the T-Check context discussed in Section 3.1. An architecture helped to determine the software requirements for the development and runtime environments. Figure 4 illustrates the system architecture designed for this T-Check investigation; the elements of the architecture are described throughout the rest of this section.

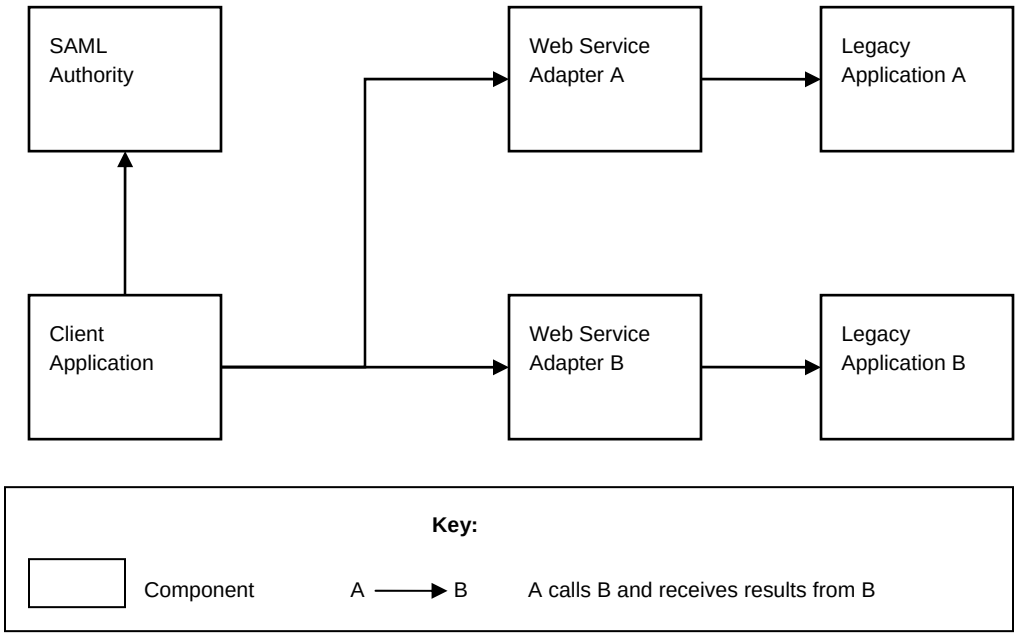

Figure 4: Notional System Architecture

In addition to the legacy applications, the architecture contains the following components:

- The SAML Authority component issues SAML authentication tokens for logged-in users.
- The client application can invoke services from the two legacy applications. It calls the SAML Authority to request authentication tokens and attaches the tokens to Web service invocations.
- The Web service Adapter A/B component exposes certain functions of the legacy applications as services. An adapter
  - validates authentication tokens received with service invocations
  - uses authentication information to authenticate to the legacy application
  - calls functions from the legacy application and returns results

Implementing one Web service is enough to investigate Web services SSO because SAML tokens contain all information necessary to validate authentication. Each additional Web service adapter can execute the same validation steps as the first adapter. Because our focus is on authentication in this T-Check investigation, we ignore details of the interaction between Web service adapter and legacy application. Instead, we implement a trivial Addition Web service that receives a number *n* and returns *n + 1*. This implementation is sufficient because authentication can be implemented independently from the actual function of the Web service.

We also assume that the runtime overhead of SSO is independent of the service. Our client application is a simple Web application that presents a form where the user can fill in a number and press a submit button. The client application then presents the result in a separate frame.

In addition, we hardcode username and password values into our client application. We know how to add authentication to the client Web application if necessary [Hunter 2001], so we do not need to add such a function to our client application. Doing so would add no value in a T-Check context, where the implemented solution is focused on evaluating the hypotheses and nothing else.

5.2 SELECTING TOOLS FOR DEVELOPMENT AND RUNTIME

One constraint in our T-Check investigation was a limited budget for the implementation. Therefore, we developed a solution using tools we were familiar with, whenever possible. This approach led us to restrict the solution to freely available tools and libraries and to use Java as the implementation language. To implement the Web services, we used the following tools:

- Apache Tomcat 6.0—a Java Servlet container to host the Web services and a simple Web client application [Apache 2007a]
- Apache Axis 1.4—a set of development tools and runtime libraries for Web services development [Apache 2005]
- Apache WSS4J 1.5.0—a Java implementation of WS-Security that supports the WS-Security SAML token profile [Apache 2006]
- Apache XML Security 1.4.1—a Java library that provides digital signatures and encryption for use in XML documents [Apache 2007b]
- OpenSAML 1.1—a Java implementation of SAML 1.0 and SAML 1.1 [Internet2 2007]
- Eclipse 3.3—a Java development environment with plug-ins that support Web services development on Tomcat and Axis [Eclipse 2007]

To host our Web services, we chose Tomcat with Axis because of previous experience using them. There are two versions of Axis, and both can be used with WS-Security. Initially we tried to use the latest version of Axis, Axis2, for Web services support and the Rampart module for WS-Security implementation. However, we could not see clearly how Rampart works with SAML tokens from the documentation and user mail list postings. Although we found code to create SAML tokens, we could not find a documented way to insert an SAML token into a SOAP message. Therefore, we decided to use Axis in combination with WSS4J. The documentation for WSS4J explicitly mentions SAML token support. Also, there is an API method that adds an

SAML token to a SOAP message. Our initial assumption was that SAML tokens needed to be created by an external tool because WSS4J does not include an SAML implementation.

For the SAML implementation, we considered the SAML 1.1 Java Toolkit and OpenSAML. We found that the Toolkit is targeted at the browser profiles specified in the SAML standard. These profiles define how a Web browser interacts with Web applications to realize an SSO solution. It is not obvious in the source code how to use the Toolkit to create an SAML token for Web service invocations. OpenSAML provides the needed functionality, so we decided to use this library instead.

The last choice we had to make was between OpenSAML version 1.1 and version 2.0. Our main concerns regarding this choice were stability and the availability of documentation. Version 1.1 was released in 2005 and seems to be stable, whereas version 2.0 is still under development. Also, there is no binary release of version 2.0 available. The only documentation for version 1.1 seems to be the Javadoc generated from the source code comments, and no user or developer guides to the library exist. Documentation for version 2.0 is available, but some sections state that the documentation is out of date. We chose to use OpenSAML 1.1 for our T-Check implementation because we considered stability more important than having the latest features. Also, we found that unit test code included in the distribution provides sufficient examples for our purposes.

Upon closer investigation, we found that WSS4J includes a class `SAMLIssuerImpl` that can be used to create SAML tokens and a class `SAMLTokenProcessor` that performs limited validation of SAML tokens. These classes use the OpenSAML library to create and process tokens.

Overall, from reading documentation, looking at provided code examples, and searching the Web for user reports and examples, we formed the impression that little guidance is available for implementing our architecture. We found articles that describe the data exchanged in a secure Web service invocation, but we found no detailed examples on how to implement such a scenario using Apache Axis.

Note that we are not recommending these tools. We chose them because we are familiar with them and they are available for free. Both of these considerations will probably be different in other projects.

5.3 UNDERSTANDING AUTHENTICATION WITH SAML TOKENS

In the following section, we describe how we used SAML authentication tokens. Our solution is based on public key cryptography with X.509 certificates for the exchange of public keys [ITU 2005]. Figure 5 shows the details of actions required by the components in the architecture to work with SAML authentication tokens.

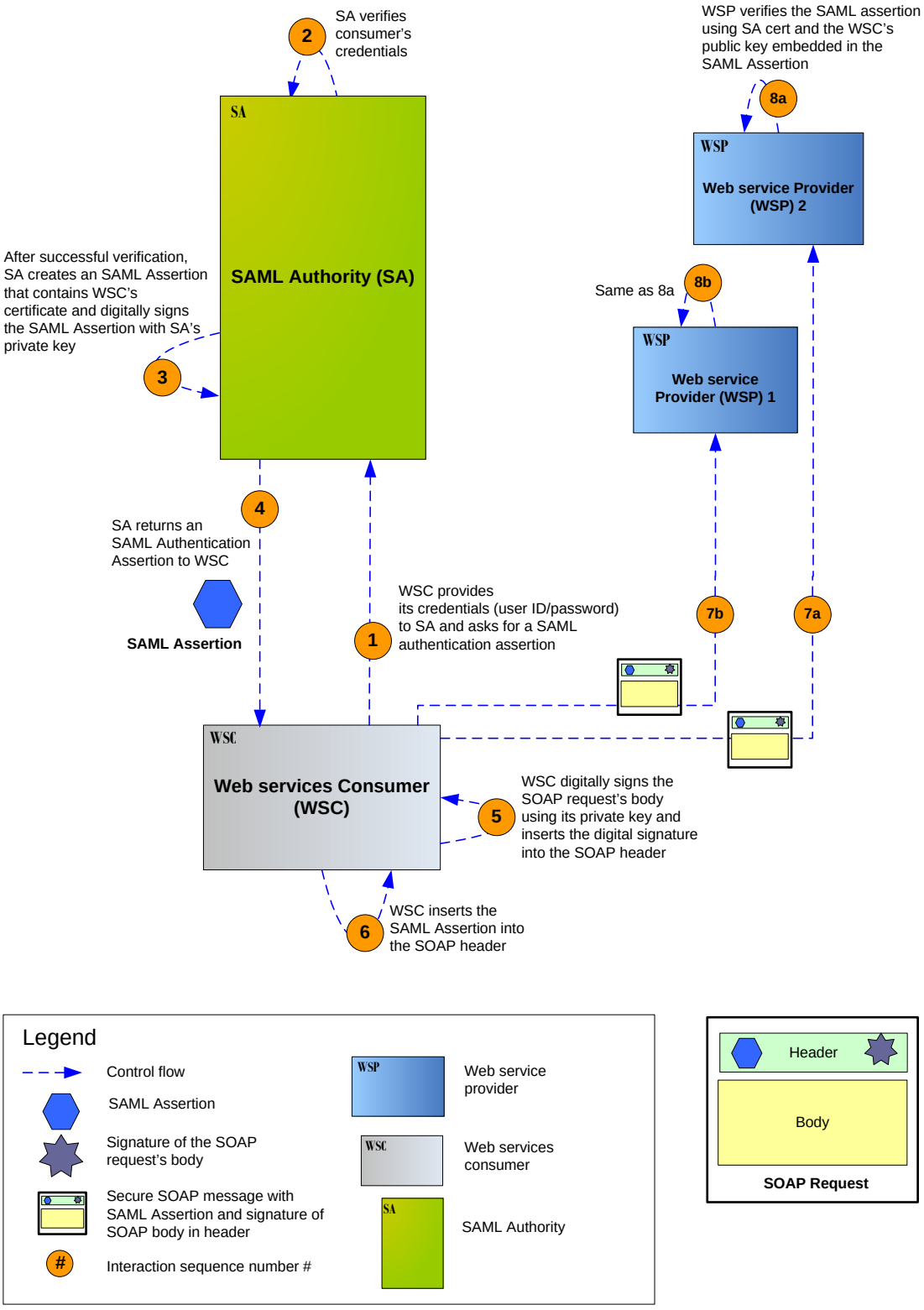

Figure 5: Steps for Using SAML Tokens

The SA attaches a digital signature to the SAML token that can be verified by the Web service. The advantage of this approach is that the Web service does not need to call the SA to verify that

the user of the client application has been authenticated. Verification of the signature is based on the X.509 certificate of the SA, which must be available to the Web services. The service provider must trust the certificate it receives from the SA. The service consumer also needs to trust the SA, because it sends the user's credentials to the SA for verification in order to retrieve an authentication token. Figure 6 shows the trust relationships that must be established, the components that need access to public and private keys, and which keys those components access. In our example T-Check context, the necessary trust relationships are already established because all components are within the same organization.

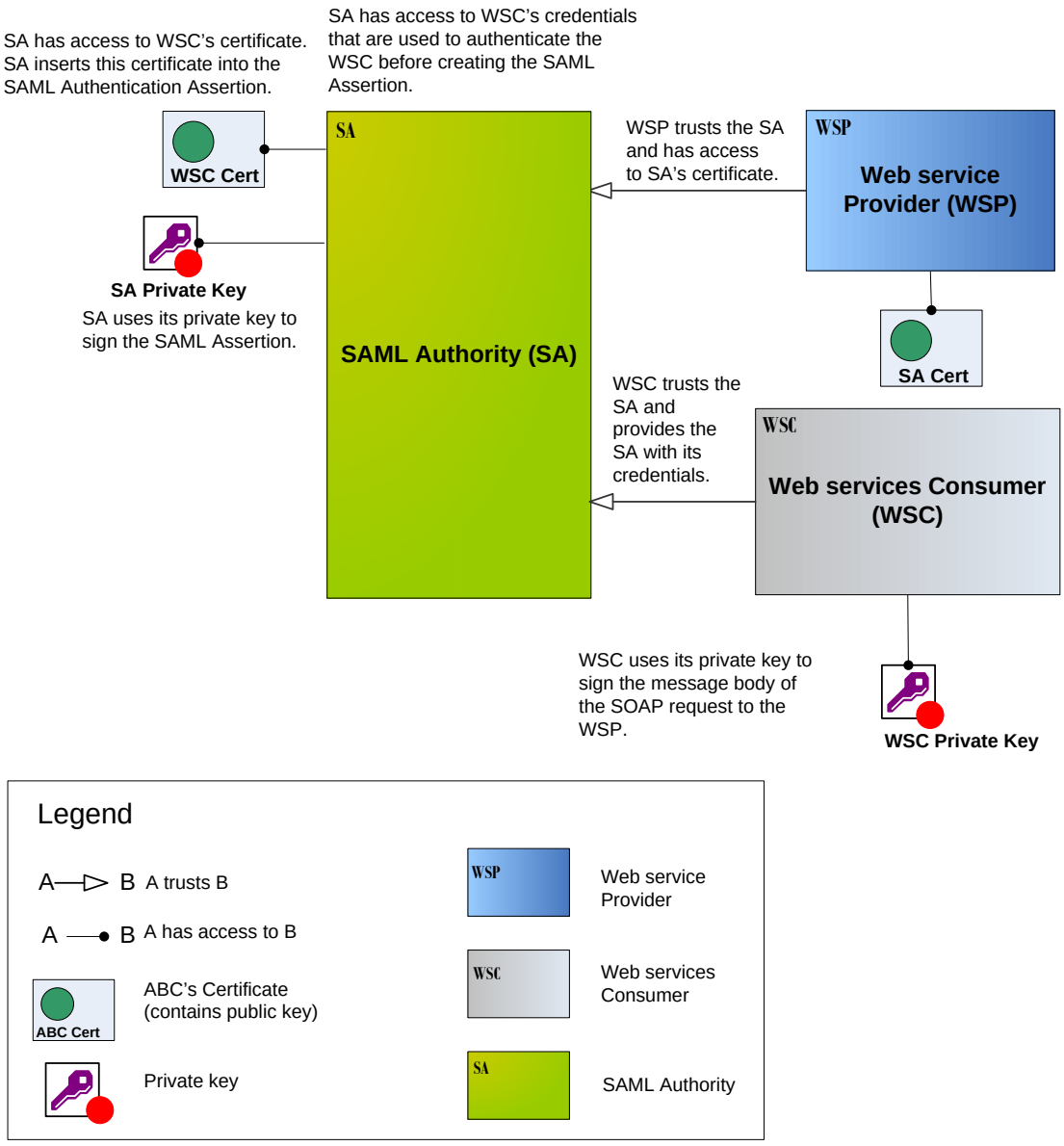

Figure 6: Trust Relationships, Certificates, and Keys

5.4 UNDERSTANDING DETAILS OF THE SAML TOKEN

We have described how SAML tokens are used in the authentication process. We now describe how those tokens work with SOAP messages. Figure 7 gives an overview of a SOAP message with an embedded SAML token. The token (SAML Assertion) is included in the SOAP header. It contains an authentication statement and a digital signature. The authentication statement's most important content in our context is the end-user's certificate that establishes the user's identity. The SAML Assertion is digitally signed by the SA, to protect the authentication statement against manipulation. This way the receiver of the SOAP message can verify the signature and be certain that the SA really issued a token for the entity whose certificate is included.

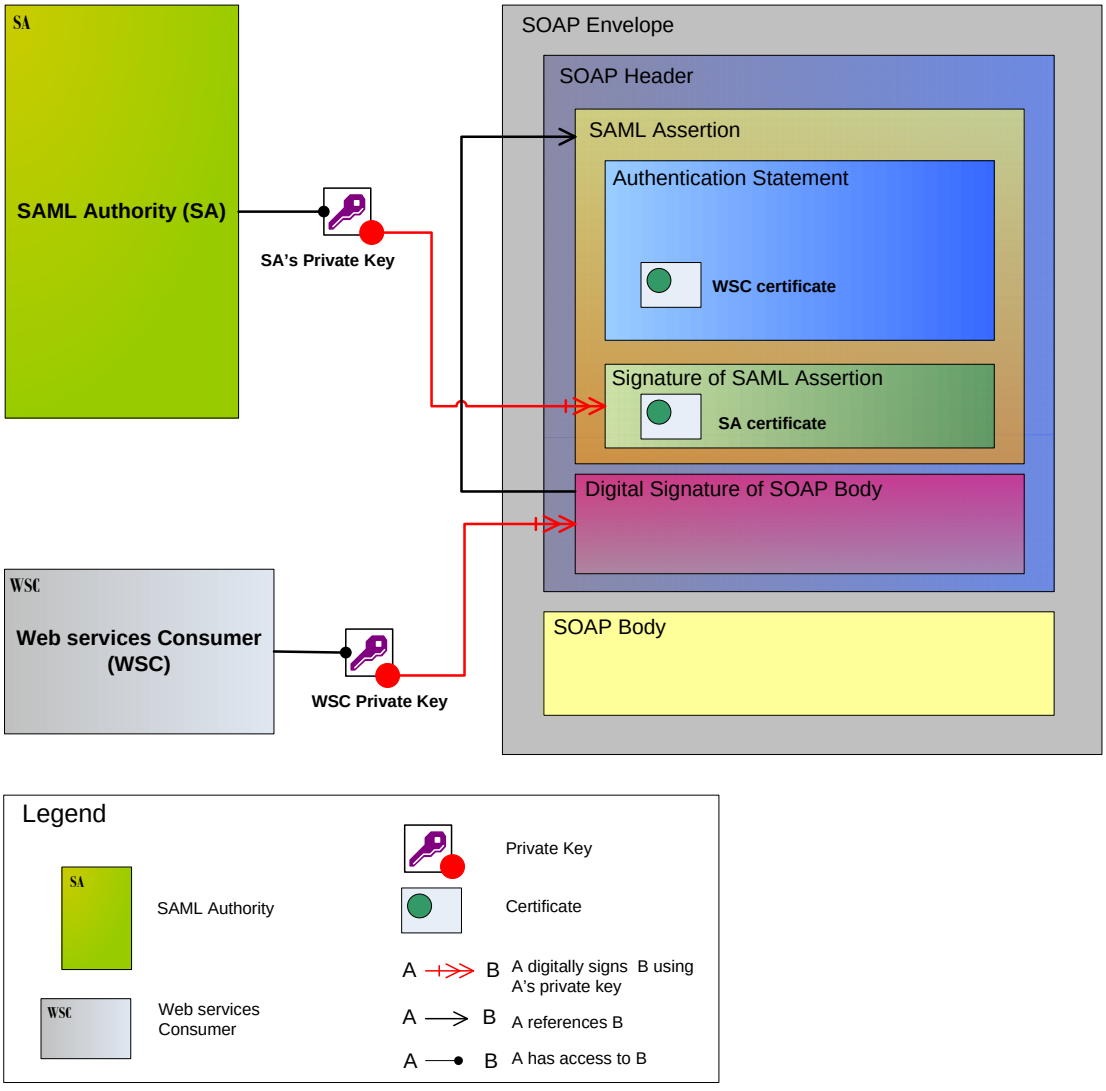

Figure 7: Overview of SOAP Message with SAML Token

The SOAP message shown in Figure 7 contains an additional level of protection in the form of a digital signature for the SOAP body. To create this signature, the client application uses the end-user's private key. The signature includes a reference to the user's certificate in the authentication statement to indicate that process. By verifying this signature, the receiver can be assured that the

SOAP body was created by the authenticated entity. Figure 8 shows a more detailed view of a SOAP message, and Table 3 explains the labeled elements.

Note that the described protection mechanisms are not effective against a replay attack, where a third party captures the SOAP message and sends it to the service provider. One way of addressing such an attack is to include a timestamp security header and to reject messages that are older than, for example, one minute.

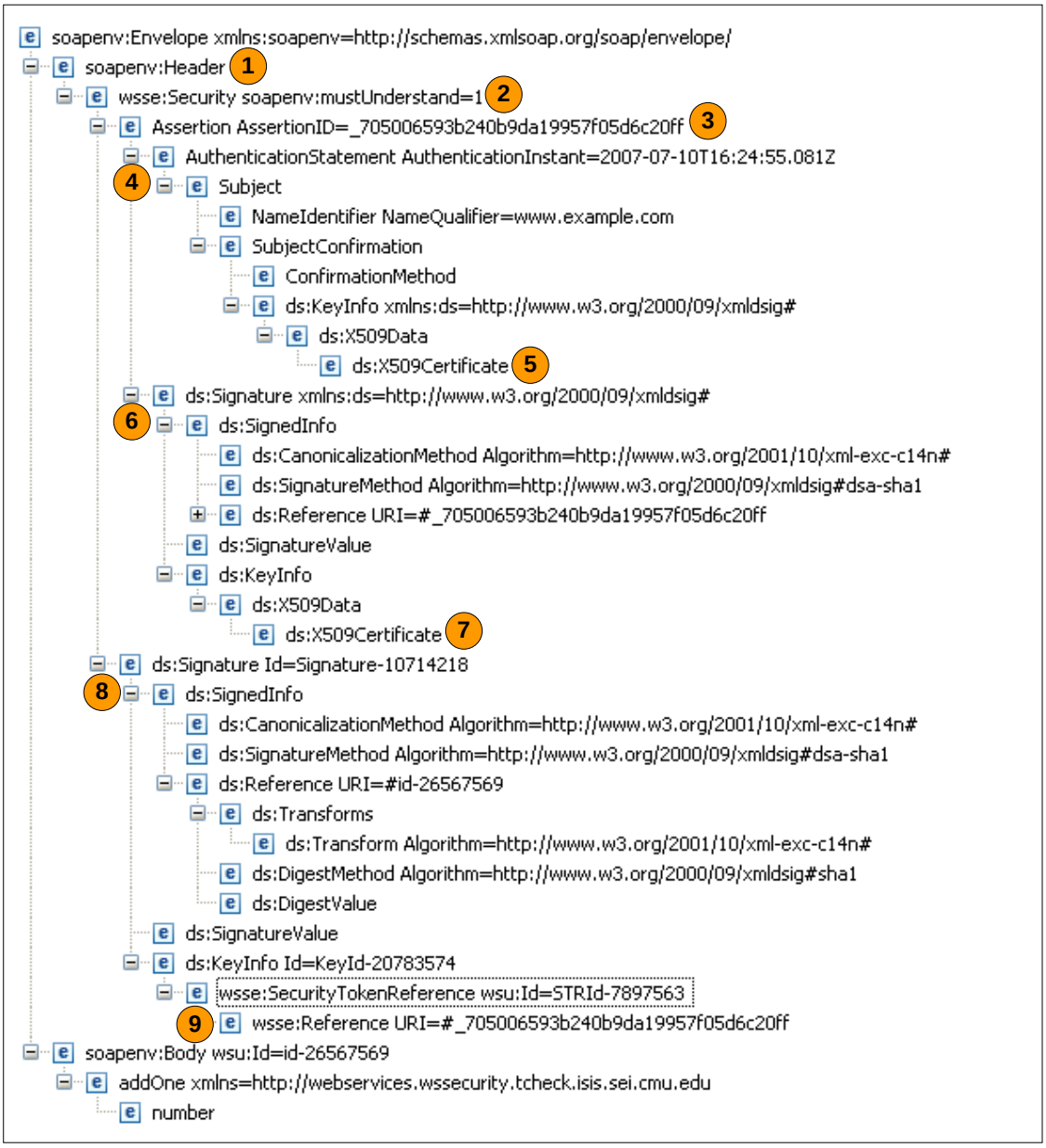

Figure 8: Detailed View of SOAP Message with SAML Token

Table 3: Elements of SOAP Message with SAML Token

Label in Figure 8	Element Description
1	SOAP Header
2	WS-Security Header
3	SAML Assertion
4	SAML Authentication Statement
5	End-user's X.509 certificate
6	Digital signature of SAML Assertion
7	SAML Authority's X.509 certificate
8	Digital signature of SOAP Body
9	Reference to end-user's X.509 certificate in SAML token

5.5 IMPLEMENTING THE T-CHECK SOLUTION

To implement our T-Check solution, we created a client application and a Web service. The client application is a simple Web application that creates a call to the Web service. We used Eclipse as our development environment and created our Web service in a bottom-up fashion. This means that we started with a Java class and used Eclipse tools to turn this class into an Axis-based Web service. The client application and Web service run on a Tomcat instance. Figure 9 shows a component and connector view of the solution; Table 4 explains the components and connector view in detail. We also include two UML sequence diagrams that show how the components process SOAP requests on the client (Figure 10) and server sides (Figure 11).

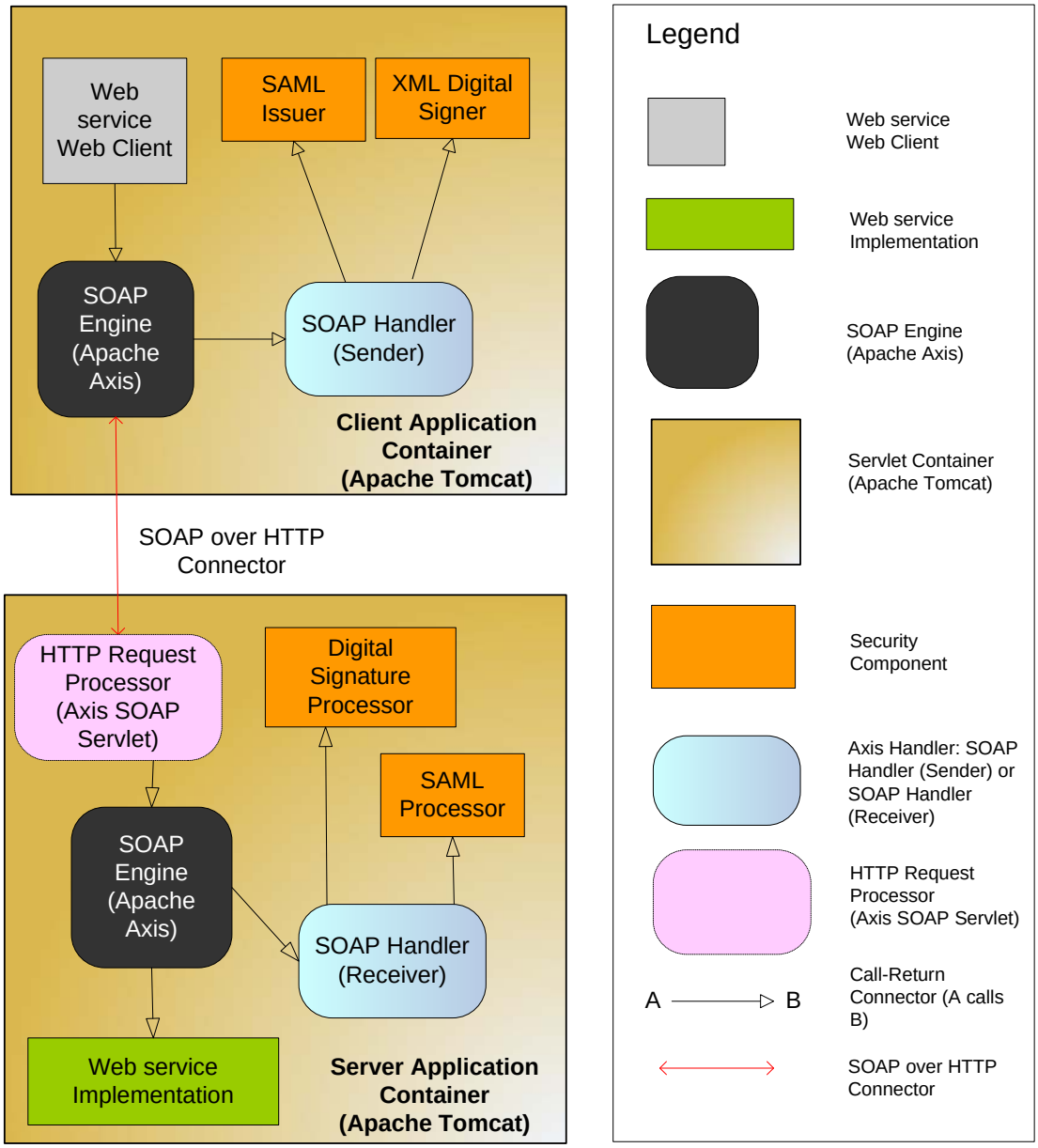

Figure 9: Component and Connector View of Architecture

Table 4: Architecture Elements and their Responsibilities

Element	Responsibility
Web service Web Client	Creates the HTML end-user interface (This component is a JSP Web client.)
SOAP Engine (Apache Axis), client side (Third-party component that can be configured to invoke custom handlers)	Creates an unsecured SOAP message (request) for invoking the Web service on the client side Passes this unsecured SOAP message to the SOAP Handler (Sender) and receives a secured SOAP message from that Handler Sends the secure SOAP message via HTTP to the Server Application Container (Apache Tomcat), where it is received by the HTTP Request Processor (Axis SOAP servlet)
SOAP Handler (Sender) (Invoked by the client-side SOAP Engine)	Creates a secured SOAP message by • extracting the user credentials from the SOAP message context • passing those credentials to the SAML Issuer • passing a SOAP message to the XML Digital Signer, after obtaining it with the SAML Assertion in the header • returning a secured and signed SOAP message to the SOAP Engine
SAML Issuer	Verifies the user credentials (user ID/password) Upon verification, the SAML Issuer creates an SAML Assertion containing the X.509 certificate for the verified user that is signed using the private key of the SAML issuer.
XML Digital Signer	Creates a digital signature of the SOAP body of the Web service request (The user's private key is used to sign the SOAP body.)
HTTP Request Processor (Axis SOAP servlet) (Deployed on the server-side Servlet Container [Apache Tomcat])	Listens for HTTP requests and forwards request received to the SOAP Engine (Apache Axis)
SOAP Engine (Apache Axis), server side	Receives the secured SOAP request from the HTTP Request Processor (Axis SOAP servlet) Invokes the SOAP Handler (Receiver) Creates a SOAP response message for the results produced by the Web service Implementation
SOAP Handler (Receiver)	Passes the secured SOAP request message to the Digital Signature Processor for validation of the SOAP body signature and to the SAML Processor for validation of the SAML Assertion
Digital Signature Processor	Receives the secured SOAP message from the SOAP Handler (Receiver) and verifies the digital signature of the SOAP message body using the X.509 certificate of the user (This certificate is part of the SOAP message header.)
SAML Processor	Validates the SAML assertion contained in the SOAP header of the message and verifies that the X.509 certificate in the SOAP body signature matches the certificate provided in the SAML Assertion
Web service Implementation	A Java class that provides the actual implementation of the Addition Web service
Client Application Container (Apache Tomcat)	Hosts the client-side components
Server Application Container (Apache Tomcat)	Hosts all server-side components

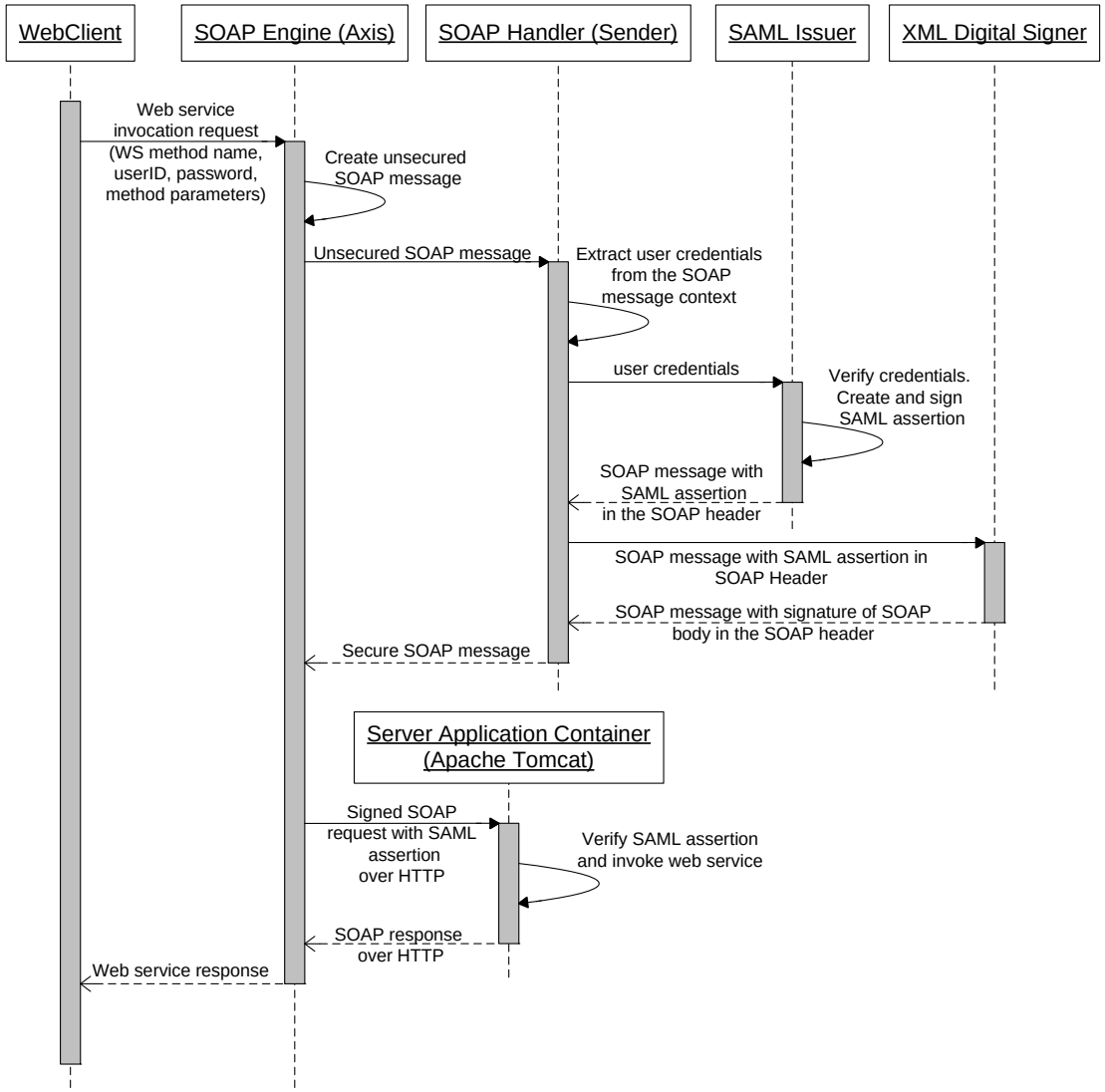

Figure 10: Client-Side Processing to Create an Outgoing SOAP Message

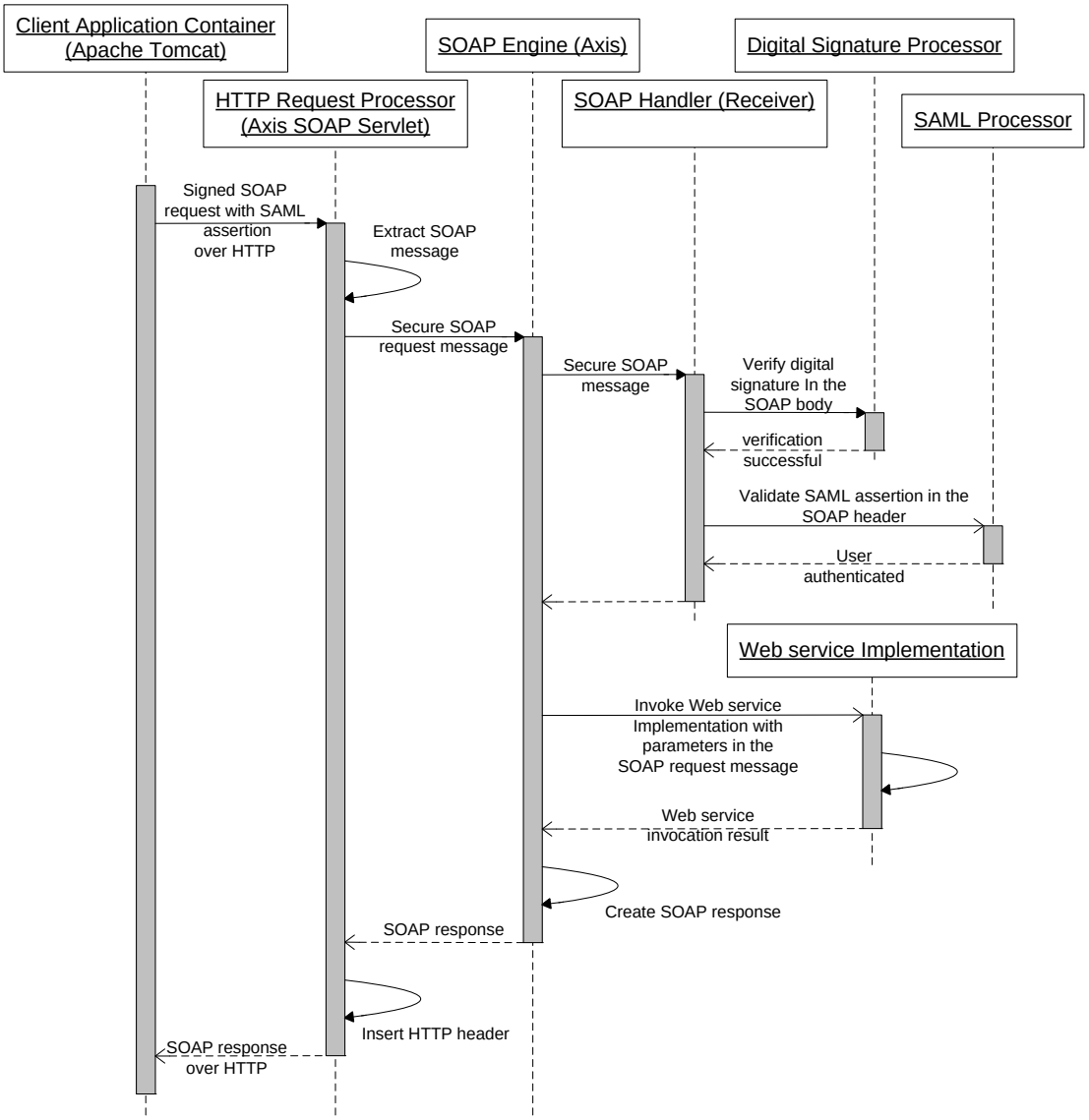

Figure 11: Server-Side Processing of an Incoming SOAP Message

All code related to basic SOAP message processing was generated. We devoted the main part of the implementation effort to configuring Axis to enable WS-Security processing with SOAP tokens, as described in Section 5.3. Overall, we developed three code modules, as shown in Figure 12. The responsibilities of the modules are described in Table 5.

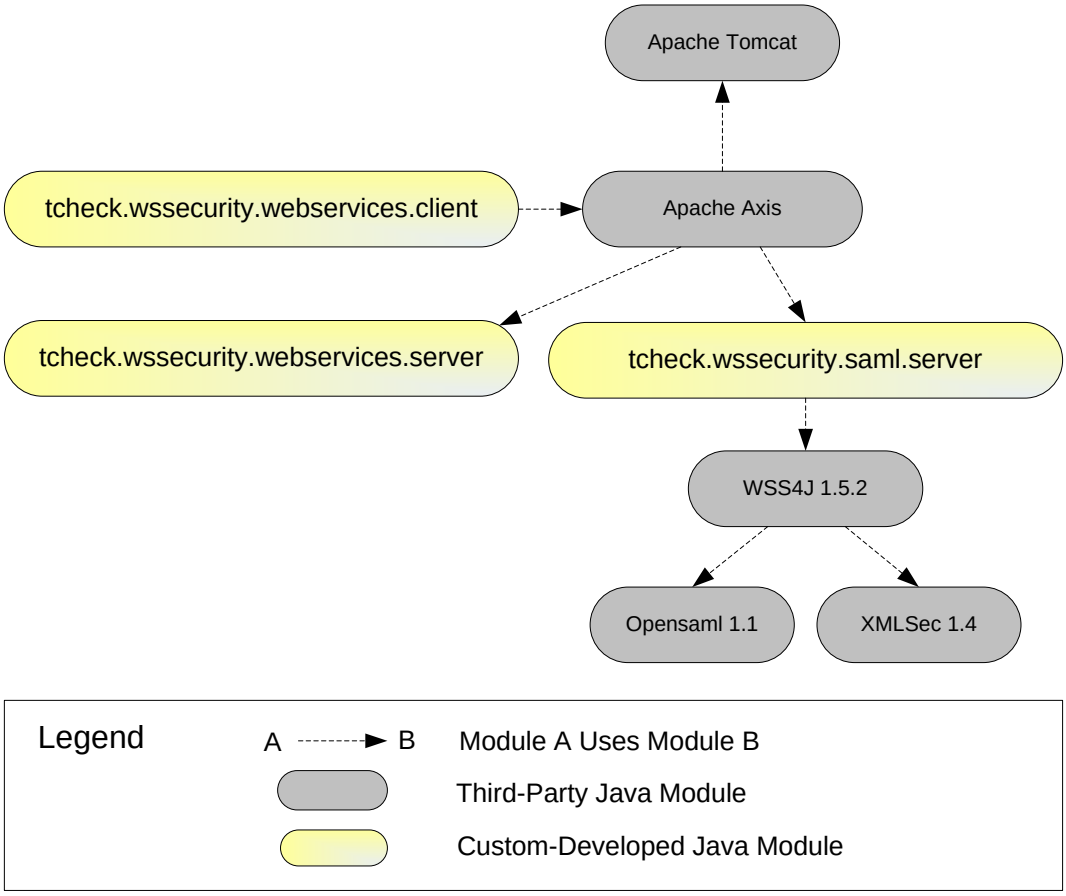

Figure 12: Module View of Architecture

Table 5: Module Descriptions

Module Name	Description
tcheck.wssecurity.webservices.client	Code for the client Web application in form of Java Server Pages
tcheck.wssecurity.webservices.server	Classes implementing the Web services functionality
tcheck.wssecurity.saml.server	Classes for validating SAML tokens received with a SOAP service request

Configuration files control many aspects of the runtime behavior of Axis. The client and server sides have their own main configuration files (client-config.wsdd and server-config.wsdd, respectively). These files define configuration details that apply to Axis and to individual Web services. For our secure Addition Web service, we configured the client side to add an SAML token to the service invocation and the server side to process it. The main configuration files reference other configuration files that control how to create digital signatures and create and process SAML tokens.

In addition to these configuration files, the application needs access to private keys and X.509 certificates. These are stored in Java key store files. We describe the configuration and key store

files in Table 6; Figure 13 depicts the relationships between them. In the Appendix, we include pertinent parts of the configuration files.

Table 6: Configuration Files

File Name	Description
Client Configuration	
client-config.wsdd	Axis configuration file for client-side SOAP request and response processing
sig-crypto.properties	Configuration of digital signature used to sign the SOAP body
saml.properties	Configuration of SAML token creation
saml-crypto properties	Configuration of digital signature used to sign the SAML Assertion
Server Configuration	
server-config.wsdd	Axis configuration file for server side request and response processing
sig-crypto.properties	Configuration of digital signature used to sign the SOAP body
Key Storage	
tcheck-keystore.jks	Public and private keys of SAML Authority and users
sa-keystore.jks	Public key of SAML Authority

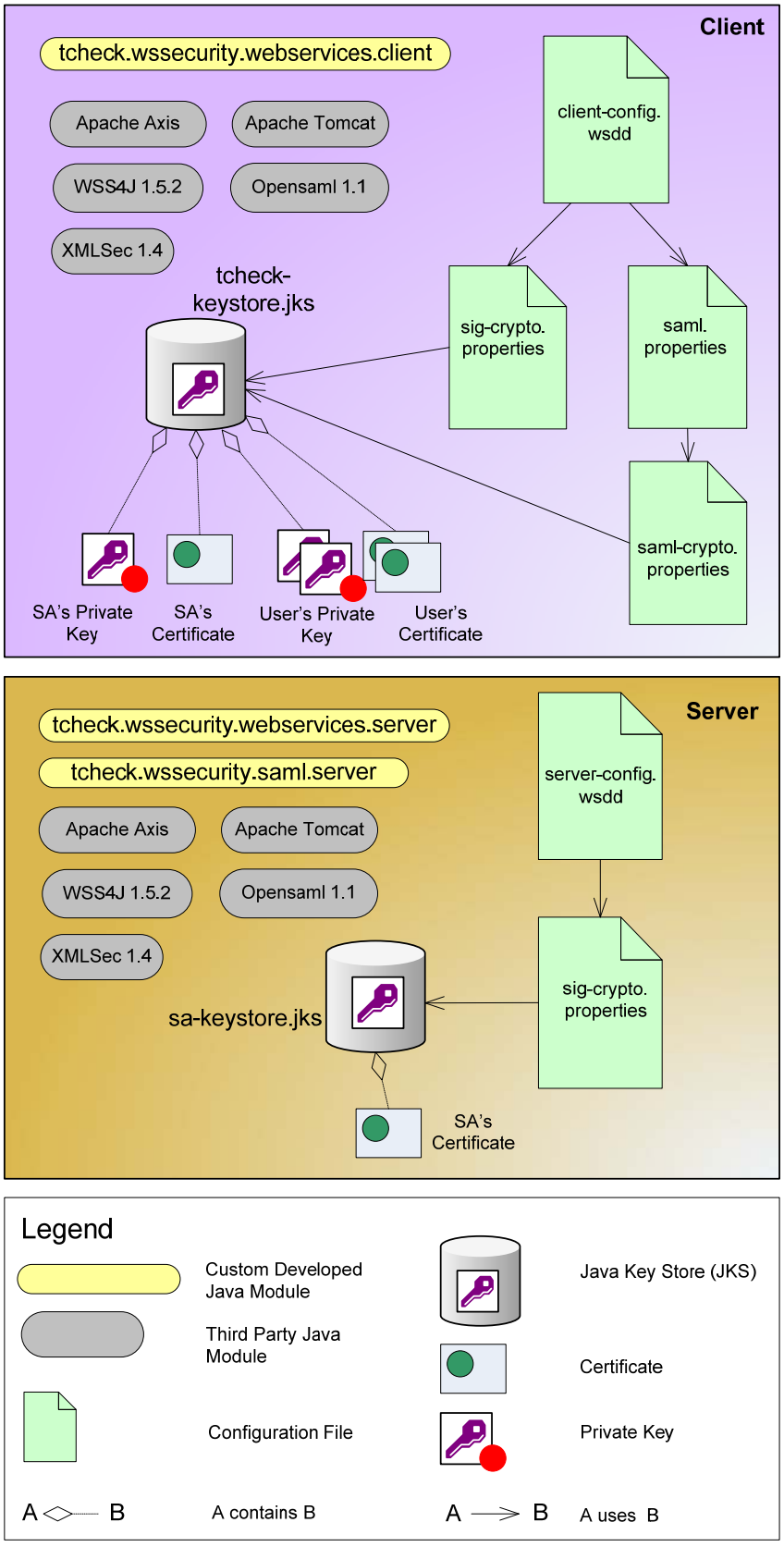

Figure 13: Deployment View of Architecture

6 Evaluation and Experiences with WS-Security in Axis

In this section, we present the results of evaluating the solution against the criteria.

6.1 RESULTS FOR HYPOTHESIS 1

Hypothesis 1: **It is possible to implement SSO for the two Web services using SAML and WS-Security.**

This hypothesis is sustained: SAML and WS-Security can be used in combination. The WS-Security SAML token profile defines how an SAML token can be embedded in the header of a SOAP message.

We also found tools and libraries that—based on available documentation—support WS-Security with SAML tokens, and we successfully used Apache Axis, Apache WSS4J, Apache XML Security, and OpenSAML to implement a basic SSO solution.

6.2 RESULTS FOR HYPOTHESIS 2

Hypothesis 2: **It is fairly easy to implement a basic SSO solution.**

This hypothesis is sustained. The effort to create the solution was approximately 10 days, which included learning about SAML and WS-Security, exploring Axis and WSS4J, and implementing and configuring them. Adding another service that uses the same authentication solution is almost trivial, because the only change that needs to be made is to a configuration file on the server side. A real-world solution will likely require more upfront effort to set up the infrastructure in which SSO can happen, but the effort to integrate a new service into an existing SSO infrastructure will be comparatively low.

The main difficulty we faced in developing SSO for our solution was the lack of documentation that explains how to use SAML tokens with WSS4J. Many articles explain how WS-Security works and how security tokens are embedded in SOAP messages. None of the articles we read explains how to use SAML tokens or set up SAML tokens with Axis and WSS4J. Also, the Axis, WSS4J, and OpenSAML documentation contains little information on this subject; we resorted to examining the source code for details on how to proceed. In addition, we had to experiment to find the correct configuration settings.

In particular, the Axis client configuration file is poorly documented. For example, Axis includes a tool that generates an initial client configuration file from the automatically generated server configuration file, but we only found out about this tool because it is mentioned in a mail list posting. Also, we tried to find out whether Axis2 provided the needed functionality. But the available documentation was almost completely useless for our purposes; so we had to abandon that option.

We discovered potential interoperability problems related to the WSS4J implementation of digital signatures. The underlying issue is that there are several ways to insert the certificate of the au-

thenticated user that the SOAP body signature needs to reference. By default, the client-side code inserts a copy of the certificate into the body signature in the SOAP header, but the server-side code expects a reference to the certificate stored elsewhere in the SOAP header. To address this situation, we added a parameter called *signatureKeyIdentifier* with the value *DirectReference* to the client-side configuration file to insert a reference to the certificate stored in the SAML token.

However, in real-world scenarios, some services may expect one form of reference and other services may expect another form. This information needs to be conveyed to service consumers to set up their configuration correctly. Further investigation into other standards (e.g., WS-SecurityPolicy) is needed to see if and how they address such issues.

6.3 RESULTS FOR HYPOTHESIS 3

Hypothesis 3: **The SSO solution will not have a major impact on the runtime behavior of the system.**

This hypothesis is sustained because the overhead introduced by the SSO solution is less than 250 ms per Web service invocation.

To measure the runtime overhead of our SSO solution, we added code to the client application that measured the time to execute 100 calls to the Addition Web service. Each call includes the generation of a new SAML token. To keep the variability of service invocation time caused by network latency low, we installed client application and Web service on the same computer. This approach is valid because the time spent on WS-Security processing is not related to network activity. Table 7 summarizes our measurements. Consequently, using WS-Security added about 84 milliseconds (ms) to each service call.

Table 7: Runtime Overhead of WS-Security

	Run #1 (100 Calls)	Run #2 (100 Calls)	Run #3 (100 Calls)	Run #4 (100 Calls)	Average per Call
No security	1078 ms	1250 ms	595 ms	890 ms	10 ms
With security	10109 ms	9375 ms	9265 ms	9030 ms	94 ms

In our T-Check scenario, an overhead of 84 ms is acceptable because the services have a comparatively long execution time. However, in high volume machine-to-machine interactions, such a performance impact may not be acceptable. For those situations, a more favorable solution would be to establish a secure session that can cover many service invocations per authentication action. Another scenario where the runtime overhead may become problematic is composed services, where a service itself calls other services during execution. If these calls need to be authenticated, the overhead can increase above an acceptable level.

When simplifying the architecture of the T-Check solution, we made the assumption that authentication is independent of the executed service (see Section 5.1). This assumption is not really valid in our implementation because the SOAP messages contain a digital signature of the message body. The time needed to calculate this signature increases with the length of the message body. Additional measurements are needed calculate the impact of a longer message on runtime

overhead. If the overhead becomes too great, it is possible to include only part of the body in the signature, limiting the amount of data included in the signature calculation.

Making a reliable statement about timing based solely on a T-Check solution is difficult because the experiment's environment, including software, hardware, and network, differs from a real-world implementation of a complete solution. We are confident, however, that the low overhead we observed in our measurements can be realized in a real system, because there are simple optimizations that can further reduce the execution time (see Table 8).

Table 8: Optimizations to Reduce Execution Time

Site	Optimization
Service consumer side	Cache security certificates in memory to save time for reloading them for every SAML token generated Reuse SAML tokens for several service invocations (The level of possible reuse depends on the validity period of the token.)
Service provider side	Cache the SAML Authority certificate in memory to save the time needed to reload it for every token validation

Implementation choices can potentially increase the time needed to validate authentication, however. One such choice for an SSO solution is to use a dedicated SOAP node to handle WS-Security authentication and other processing. There is a tradeoff between execution time and the benefits of centralized handling of authentication decisions.

Also, SAML creation will likely be different in a real-world implementation. In this T-Check investigation, we stored certificates in a Java key store file; in a real system, a directory service would likely be used to store this information in a central repository.

6.4 RESULTS FOR HYPOTHESIS 4

Hypothesis 4: **The SSO solution can provide the required access control.**

Although we did not implement access control in our example solution, we see a simple way of adding such functionality. The SAML token in our solution contains only an authentication statement. To add access control, we can also include information about the user's permissions in the token. The set of SAML statements includes attribute statements that can convey information about subjects. If we add an attribute statement that names the user's role, the Web service can evaluate this role to grant access to the service.

There only needs to be agreement about the meaning of roles. In the T-Check scenario, there are two roles:

1. *ALL*—a user in this role has access to legacy systems A and B.
2. *ONE*—a user in this role has access only to legacy system B.

In a real-world implementation, the source of role information would be a directory service that also contains the users' encryption keys and the X.509 certificates used to create digital signatures.

7 Future Work

In the future, we plan to work on some areas we could not cover during this T-Check investigation. Those areas are

- commercial/proprietary tools

  Often, organizations acquire Web service infrastructure components, such as an enterprise service bus. These commercial components support security and include development and management tools. We would like to compare such an environment with the open source environment we used in this T-Check investigation.

- more WS-* standards

  The WS* landscape is so complex that we had to limit out T-check investigation to a subset of these standards. We think it would be worthwhile to conduct further investigation into WS-* standards, beginning with WS-Trust and WS-SecurityPolicy.

- more complex SSO scenarios

  We would like to implement more complex SSO scenarios in order to validate the hypotheses described in this report on those scenarios. In particular, we would like to experiment with federated identity management to gain experience with the resulting interoperability challenges.

8 Conclusions and Call for Response

Our T-Check investigation into SAML and WS-Security shows that current standards can be used in a basic SSO implementation. Although the learning curve for the standards and tools was steep, we feel that SSO in a Web services environment has great potential because it can be used easily by the developer and is transparent to the end user. Overall, this T-Check investigation greatly contributed to our understanding of how security can be addressed in a Web services context.

The team in the Integration of Software-Intensive Systems (ISIS) Initiative at the SEI that is investigating SSO and other technologies using the T-Check approach is interested in feedback from and collaboration with the communities that are considering technologies for service-oriented environments. Write to the ISIS team at isis-sei@sei.cmu.edu.

Appendix Axis Configuration Files

Client-Side Configuration Files

Configuration entries relevant to WS-Security are highlighted.

client-config.wsdd File

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
    <globalConfiguration>
        <parameter name="disablePrettyXML" value="true" />
        <parameter name="enableNamespacePrefixOptimization" value="true" />
    </globalConfiguration>

    <service name="AdditionWebService" provider="java:RPC"
           style="wrapped" use="literal">
        <operation name="addOne" qname="ns1:addOne" returnQName="ns1:addOneReturn"
              returnType="xsd:int" soapAction=""
              xmlns:ns1="http://webservices.wssecurity.tcheck"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <parameter qname="ns1:number" type="xsd:int" />
        </operation>
        <parameter name="allowedMethods" value="addOne" />
        <parameter name="wsdlPortType" value="AdditionWebService" />
        <parameter name="typeMappingVersion" value="1.2" />
        <parameter name="schemaQualified"
              value="http://webservices.wssecurity.tcheck " />
        <parameter name="wsdlServicePort" value="AdditionWebService" />
        <parameter name="className"
              value="tcheck.wssecurity.webservices.AdditionWebService" />
        <parameter name="wsdlTargetNamespace"
              value="http://webservices.wssecurity.tcheck" />
        <parameter name="wsdlServiceElement" value="AdditionWebServiceService" />
        <requestFlow>
            <handler type="java:org.apache.ws.axis.security.WSDoAllSender">
                <parameter name="action" value="SAMLTokenSigned" />
                <parameter name="samlPropFile" value="saml.properties" />
                <parameter name="signaturePropFile" value="sig-crypto.properties"/>
                <parameter name="signatureKeyIdentifier" value="DirectReference" />
            </handler>
        </requestFlow>
    </service>
    (...)
</deployment>
```

sig-crypto.properties File

```
org.apache.ws.security.crypto.provider=
       org.apache.ws.security.components.crypto.Merlin
org.apache.ws.security.crypto.merlin.file=tcheck-keystore.jks
org.apache.ws.security.crypto.merlin.keystore.type=JKS
```

```
org.apache.ws.security.crypto.merlin.keystore.password=keystore
```

saml.properties File

```
org.apache.ws.security.saml.issuerClass=
       org.apache.ws.security.saml.SAMLIssuerImpl
org.apache.ws.security.saml.issuer.cryptoProp.file=saml-crypto.properties
org.apache.ws.security.saml.issuer.key.name=isisap
org.apache.ws.security.saml.issuer.key.password=isisappass
org.apache.ws.security.saml.issuer=isis.sei.cmu.edu
org.apache.ws.security.saml.subjectNameId.name=
       uid=joe,ou=people,ou=saml-demo,o=example.com
org.apache.ws.security.saml.subjectNameId.qualifier=www.example.com
org.apache.ws.security.saml.authenticationMethod=password
org.apache.ws.security.saml.confirmationMethod=keyHolder
```

saml-crypto.properties File

```
org.apache.ws.security.crypto.provider=
       org.apache.ws.security.components.crypto.Merlin
org.apache.ws.security.crypto.merlin.file=tcheck-keystore.jks
org.apache.ws.security.crypto.merlin.keystore.type=JKS
org.apache.ws.security.crypto.merlin.keystore.password=keystore
org.apache.ws.security.crypto.merlin.keystore.alias=isisap
```

Server-Side Configuration Files

server-config.wsdd File

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns=http://xml.apache.org/axis/wsdd/
          xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
    <globalConfiguration>
        <parameter name="disablePrettyXML" value="true" />
        (...)
    </globalConfiguration>
    (...)
    <service name="AdditionWebService" provider="java:RPC"
           style="wrapped" use="literal">
        <operation name="addOne" qname="ns2:addOne" returnQName="ns2:addOneReturn"
               returnType="xsd:int" soapAction=""
               xmlns:ns2="http://webservices.wssecurity.tcheck"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <parameter qname="ns2:number" type="xsd:int" />
        </operation>
        <parameter name="allowedMethods" value="addOne" />
        <parameter name="typeMappingVersion" value="1.2" />
        <parameter name="wsdlPortType" value="AdditionWebService" />
        <parameter name="className"
               value="tcheck.wssecurity.webservices.AdditionWebService" />
        <parameter name="wsdlServicePort" value="AdditionWebService" />
        <parameter name="schemaQualified"
               value="http://webservices.wssecurity.tcheck" />
        <parameter name="wsdlTargetNamespace"
               value="http://webservices.wssecurity.tcheck" />
        <parameter name="wsdlServiceElement" value="AdditionWebServiceService" />
```

```
        <requestFlow>
            <handler type="java:tcheck.wssecurity.saml.SAMLValidationHandler">
                <parameter name="action" value="Signature SAMLTokenSigned" />
                <parameter name="signaturePropFile" value="sig-crypto.properties"/>
                <parameter name="signatureKeyIdentifier" value="DirectReference" />
            </handler>
        </requestFlow>
    </service>
(...)
</deployment>
```

sig-crypto.properties File

```
org.apache.ws.security.crypto.provider=
       org.apache.ws.security.components.crypto.Merlin
org.apache.ws.security.crypto.merlin.file=sp-keystore.jks
org.apache.ws.security.crypto.merlin.keystore.type=JKS
org.apache.ws.security.crypto.merlin.keystore.password=keystore
```

References

URLs are valid as of the publication date of this document.

[Apache 2005]
The Apache Software Foundation. *Web Services - Axis*. http://ws.apache.org/axis/ (2000−2005)

[Apache 2006]
Apache Web Services. *Apache WSS4J*. http://ws.apache.org/wss4j/ (2004−2006)

[Apache 2007a]
The Apache Software Foundation. *Apache Tomcat*. http://tomcat.apache.org/ (1999−2007)

[Apache 2007b]
The Apache Software Foundation. *Welcome to XML Security*. http://xml.apache.org/security/ (2007)

[Centrify 2007]
Centrify Corporation. *DirectControl's Integrated Support for Microsoft ADFS*. http://www.centrify.com/directcontrol/adfs.asp (2004−2007)

[Eclipse 2007]
Eclipse. *Eclipse – an open development platform*. http://www.eclipse.org/ (2007)

[Hunter 2001]
Hunter, Jason. *Java Servlet Programming, 2nd Edition.* O'Reilly, 2001.

[IBM 2002]
IBM. *Adapting legacy applications as Web services*. http://www.ibm.com/developerworks/webservices/library/ws-legacy/ (2002)

[IBM 2004]
IBM. Web Services Security. http://www.ibm.com/developerworks/library/specification/ws-secure/ (2004)

[IBM 2006]
IBM. *Web Services Federation Language. Version 1.1*. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-fed/WS-Federation-V1-1B.pdf (2006)

[Internet2 2007]
Internet2. *OpenSAML - an Open Source Security Assertion Markup Language implementation*. http://www.opensaml.org/ (2007)

[ITU 2005]
International Telecommunication Union. *Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate framework*. (ITU Recommendation X.509). http://www.itu.int/itudoc/itu-t/aap/sg17aap/history/x509/index.html (2005)

[Lewis 2005]
Lewis, Grace A. & Wrage, Lutz. *A Process for Context-Based Technology Evaluation* (CMU/SEI-2005-TN-025, ADA441251). Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/publications/documents/06.reports/05tn025.html

[Lewis 2006]
Lewis, Grace A., & Wrage, Lutz. *Model Problems in Technologies for Interoperability: Web Services* (CMU/SEI-2006-TN-021, ADA454363). Software Engineering Institute, Carnegie Mellon University, 2006. http://www.sei.cmu.edu/publications/documents/06.reports/06tn021.html

[Liberty 2007]
Liberty Alliance Project. *Specifications*. http://www.projectliberty.org/liberty/resource_center/specifications (2007)

[Lockhart 2005]
Lockhart, Harold. *Demystifying SAML*. http://dev2dev.bea.com/pub/a/2005/11/saml.html (2005)

[Microsoft 2002]
Microsoft Corporation. *Security in a Web Services World: A Proposed Architecture and Roadmap*. http://msdn2.microsoft.com/en-us/library/ms977312.aspx (2002)

[Microsoft 2007]
Microsoft Corporation. *Active Directory Federation Services (ADFS)*. http://technet2.microsoft.com/WindowsServer/en/library/050392bc-c8f5-48b3-b30e-bf310399ff5d1033.mspx?mfr=true (2007)

[OASIS 2005]
Organization for the Advancement of Structured Information Standards. *OASIS UDDI*. http://www.uddi.org/ (2005)

[OASIS 2006a]
Organization for the Advancement of Structured Information Standards. *Web Services Security: SOAP Message Security 1.1*. http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf (2006)

[OASIS 2006b]
Organization for the Advancement of Structured Information Standards. *SAML 1.1 Specification*. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security#samlv11 (2006)

[OASIS 2006c]
Organization for the Advancement of Structured Information Standards. *Web Services Security: SAML Token Profile 1.1*. http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf (2006)

[OASIS 2007a]
Organization for the Advancement of Structured Information Standards. *WS-SecureConversation 1.3 OASIS Standard*. http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-secureconversation-1.3-os.html (2007)

[OASIS 2007b]
Organization for the Advancement of Structured Information Standards. *WS-SecurityPolicy 1.2 Committee Specification.* http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-cs.html (2007)

[OASIS 2007c]
Organization for the Advancement of Structured Information Standards. *WS-Trust 1.3. OASIS Standard*. http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html (2007)

[Oracle 2007]
Oracle Corporation. *Oracle Phaos*. http://www.oracle.com/technology/products/id_mgmt/phaos/index.html (2007)

[Rosenberg 2004]
Rosenberg, J., Remy D. *Securing Web Services with WS-Security: Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption.* SAMS Publishing, 2004.

[Rouault 2005]
Rouault, Jason. *Making sense of the federation protocol landscape*. http://devresource.hp.com/drc/resources/fed_land/federation_landscapeHP.pdf (2005)

[SourceID 2006]
SourceID. *SAML 1.1 Java Toolkit*. http://www.sourceid.org/projects/saml_1_1_toolkit (2006)

[SourceID 2007]
SourceID. *WS-Federation for Apache 2.0 Toolkit Overview*. http://www.sourceid.org/projects/ws-federation-apache (2007)

[Wallnau 2001]
Wallnau, Kurt, Hissam, Scott, & Seacord, Robert. *Building Systems from Commercial Components*. Addison-Wesley, 2001.

[Wikimedia 2006a]
Wikimedia Foundation. *Security Assertion Markup Language*. http://en.wikipedia.org/wiki/Security_Assertion_Markup_Language (2006)

[Wikimedia 2006b]
Wikimedia Foundation. *Single Sign On*. http://en.wikipedia.org/wiki/Single_sign-on (2006)

[Wikimedia 2006c]
Wikimedia Foundation. *WS-Security*. http://en.wikipedia.org/wiki/WS-Security (2006)

[W3C 2003]
World Wide Web Consortium. *HTTP - Hypertext Transfer Protocol*. http://www.w3.org/Protocols/ (2003)

[W3C 2004]
World Wide Web Consortium. *Web Services Architecture*. http://www.w3.org/TR/ws-arch/ (2004)

[W3C 2005]
World Wide Web Consortium. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Working Draft 3 August 2005*. http://www.w3.org/TR/wsdl20/ (2005)

[W3C 2006]
World Wide Web Consortium. *Web Services Policy 1.5 – Primer*. http://dev.w3.org/cvsweb/2006/ws/policy/ws-policy-primer.html?rev=1.9 (2006)

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	December 2007	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
T-Check in Technologies for Interoperability: Web Services and Security—Single Sign-On	FA8721-05-C-0003

6. AUTHOR(S)
Lutz Wrage, Soumya Simanta, Grace A. Lewis, Saul Jaspan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER
CMU/SEI-2008-TN-026

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQ ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

A single sign-on (SSO) solution is intended to provide a single authentication point for a set of Web services. The SSO solution forwards the necessary authentication information to the Web services, which in turn authenticate the end user to legacy systems that implement the Web services' functionality. This technical note presents the results of applying the T-Check approach in an initial investigation of two Web services standards, WS-Security and SAML, to create an SSO solution that works inside a single organization. This approach involves (1) formulating hypotheses about the technology and (2) examining these hypotheses against specific criteria through hands-on experimentation. The outcome of this two-stage approach is that the hypotheses are either fully or partially sustained or refuted. In this report, four hypotheses—based on claims found in experience reports and on vendor Web sites—are examined: (1) it is possible to implement SSO for the two Web services using SAML and WS-Security; (2) it is fairly easy to implement a basic SSO solution; (3) the SSO solution will not have a major impact on the runtime behavior of the system; and (4) the SSO solution can provide the required access control. The first three hypotheses were sustained; it was not necessary to implement the fourth one to list options for adding access control.

14. SUBJECT TERMS	15. NUMBER OF PAGES
T-Check, Tcheck, single sign on, security, Web Services, Web service, SAML, WS-Security, service-orientation, SOA, service-oriented architecture, model problem, interoperability	52

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL